# Modeling Security Protocols Using UML 2

Sandra Smith, Alain Beaulieu and W. Greg Phillips

Department of Electrical and Computer Engineering
Royal Military College of Canada
Kingston, Ontario, Canada, K7K 7B4
{sandra.smith,alain.beaulieu,greg.phillips}@rmc.ca

**Abstract.** Security protocols must be designed to ensure the integrity of electronic communications between participants. Although the design of secure communication protocols has improved over the years the tasks of building and validating these protocols remain inherently difficult. Security protocols may fail due to unintended use, malicious attacks, incorrect logic or incorrect transition from design to code. We present our research to investigate the use of UML 2 to model, verify and validate security protocols.

**Key words:** Modeling, security protocols, UML 2, Model Driven Development

## 1 Introduction

One traditional approach to the verification of a security protocol involves the creation of a protocol model and the analysis of that model to determine its security properties. We can test the model, explore its state-space using a model checker, or prove theorems about it in order to assure ourselves that the protocol under verification has the desired properties. However, even if the model is found to be well designed, there is still a gap to be traversed from the model to its implementation; this is a source of potential security flaws.

Version 2 of the Unified Modeling Language standard [1] introduces several constructs that directly support what has come to be called "model driven development" (MDD): the automatic or semi-automatic generation of implementations from relatively high-level models. In our current work we are exploring the extent to which these UML 2 constructs (hierarchically nested capsules, typed ports, messages, and behaviour-specifying state machines) support the modeling and verification of security protocols. Our work is still in its early stages; however, we have already seen indications that the approach holds promise.

In this paper we provide some background to the problem, discuss related approaches, introduce the key UML 2 features of interest, and report on our initial modeling experiences.

## 2 Motivation and Background

One of the key considerations in systems development today is the requirement to integrate security in order to incorporate defenses to protect against malicious attacks. Due to lack of expertise with systems developers, security is not always considered as part of the initial requirements and design but instead is something that is engineered into the software after the design work is complete and functional requirements have been captured and modeled. This results in the actual definition and integration of the security policy being treated separately and, generally, after the fact, meaning that security policy enforcement mechanisms are retro-fitted to pre-existing designs [2].

This late integration of design and models for security and systems requirements, translates into an increased risk of introducing vulnerabilities that can be exploited by an external threat. The vulnerabilities encountered within a secure system are often a result of flaws within the implementation as opposed to flaws in the actual design of the security mechanism [3]. Given that attacks on systems can pose a serious threat to economic and physical well-being of people and organizations, any effort directed at improving quality in the development of secure systems can aid in reducing the risk and exposure to these types of attacks [4].

In order to mitigate the exposure to attacks on systems, security protocols are employed as a mechanism to prescribe the sequence of interactions that can occur between principals within the system in order to achieve a certain end [5]. The protocol is basically a set of rules and conventions that outline the communication framework between principals, where principals can be end-users, processes or computer systems [6]. To establish secure communications over insecure open networks or distributed systems, security protocols make use of cryptographic mechanisms where at least part of one message is encrypted. Protocols that employ encryption are often referred to as cryptographic protocols, for ease of use within this paper the term security protocol will be used to refer to both types of protocols.

In order to effectively incorporate mechanisms such as security protocols early in the systems development process, a mechanism is needed to support better understanding, design and implementation by system developers who may not be specialists in security engineering. As observed by Roger Needham, "security protocols are three line programs that people still manage to get wrong" [1]. Modeling provides a means to integrate security engineering with software engineering, by better management of complexity through abstraction and the ability to visualize the entire system. In order for a modeling language to be successfully applied to both security and software engineering it needs to meet several basic criteria:

[1] This quote attributed to Needham is often used by authors, although we could not find the original source.

– It must be usable in that modeling a system and stating its properties should be easy to learn for software practitioners and not require a high degree of expertise or be a barrier to adoption;
– It must be comprehensible in that it presents information about a software system in an format that can be understood by non-practitioners;
– It needs to be expressive enough to present all of the key concepts of the system without losing any important details;
– It must be predictive in that the model can be analyze or executed in order to reveal non-obvious properties of the system;
– It must support the effective transition to implementation such that it provides the means to support the accurate translation of the design into code; and
– It must be cost-effective in that the model is less expensive to build and analyze than the software system [7, 8].

Any proposed approach must take into account industry standards for the modeling of computer systems and, as such, the Unified Modeling Language (UML) exists as a de-facto standard for object-oriented modeling. With the increasing use of UML, there are opportunities to integrate security policy development into the software development process through the unification of system and security models under a single modeling language [4].

There has been a body of research examining the use of UML for modeling security protocols. To date the solutions have focused on extending UML to provide direct support for the security domain. Several different variations on extensions to UML do exist, but no single one has emerged as a de-facto standard. In addition, these extensions have not yet taken advantage of the support that UML 2.0 provides to Model Driven Development (MDD). As well any extension to UML requires security modeling experts to learn new notations to effectively communicate design intent. The problem addressed within this body of work is to determine whether the Unified Modeling Language (UML) version 2.0, as defined, can adequately model security protocols.

## 3 Related Work

Numerous approaches have addressed the problem of integrating the modeling of security and security protocols within the systems development lifecycle. We discuss these methods based on the modeling criteria elaborated above.

### 3.1 Formal Methods - Communicating Sequential Processes

In order to mitigate the exposure to attacks on secure systems, it is often suggested that the use of formal methods will ensure the quality of the system from design to verification. Formal methods represent a rigorous mathematical analysis approach that typically involve proofs of correctness. The use of formal methods supports the understanding of the system so that it is precise enough to make any conclusions drawn about the system more reliable than they otherwise might be. As an example, the *Communicating Sequential Processes (CSP)* is a process algebra that provides a mathematical framework for describing systems where there are parallel agents that interact through the exchange of messages [5]. Since security properties and security protocols are concerned with the flow of messages within a network, CSP provides a mechanism to describe and analyze these protocols [9].

The strength of the CSP approach is that it provides a strong expressive framework to construct a provable mathematical model of a system and conduct a formal study of that system in order to reason about security protocols and their properties. This is important in verifying the correctness of the security protocol design. However, this approach fails in two major ways:

1. It is not specifically suited to the broader issue of integrating security engineering into software engineering in order to bridge the gap between design and implementation; this gap also extends to design-code verification; and
2. While formal methods may be expressive and predictive, they have not been widely adopted by industry given that they require more specialized knowledge and training to effectively apply them. Unfortunately, this makes these types of approaches less usable and cost-effective then other methods as practitioners and non-practitioners cannot easily learn and apply these methods.

### 3.2 Custom Visual Formalisms

Some solutions present visual modeling formalisms intended to address both the complexity of formal methods and the deficiencies perceived to exist within other modeling languages. In a recent paper, J. McDermott contends that existing approaches to modeling are insufficient to support the visual modeling of security protocols [10]. McDermott presents an alternative visual modeling formalism referred to as GSPML to meet the specific needs of security protocol modeling.

GSPML, which does not stand as an acronym for any given name, is intended as a solution to the security-specific problem of modeling protocols using a visual modeling language. A key feature of GSPML is that it provides a notation that can capture all traces of a security protocol within a single model in order to reveal any bad traces that may introduce vulnerabilities into the system [10]. The strength of GSPML is that it uses a visual models to represent complex concepts in order to aid in understanding and verification. While intended for use by security specialists, GSPML is also intended to provide a bridge from the specialist to software developers given its visual presentation approach [10]. However the method fails in that:

1. It is specific to security protocols and it introduces a new language and notation that is targeted toward security specialists and not specifically intended for use by non-practitioners;
2. It does not integrate security design with software engineering; and

3. There is no specification or tool for rigorous, traceable transition from model to code. As well there is no verification and validation method specified by the authors to use the model to ensure that the implementation is correct. The proposed model is not executable and cannot produce auditable traces. This may leave a significant security gap between design and implementation.

### 3.3 Extensions to UML

There are a number of approaches to modeling security protocols that extend UML in order to create profiles specific to the security domain. These UML extensions take into consideration the concern of how to effectively address integrating security as a component of the overall system design in order to reduce the risk of introducing vulnerabilities during implementation. UMLsec is an approach that formally specifies security requirements through stereotypes, tagged values and constraints to model security requirements such as secure links, data security, critical and fair exchange, to name only a few [4]. SecureUML is an extension for specifying access control policies for actions on protected resources through the use of a security modeling language where the abstract syntax and semantics allow it to be combined with design modeling languages other than UML [11]. The UML extension for security protocol (USP) is a framework that encapsulates knowledge for modeling of security protocols through two separate profiles, one representing physical organization and the other behaviours, that are intended to be used by developers that may not be specialists in security [6].

The extensions to UML described above all provide mechanisms for developers who may not be experts in security in order to integrate security requirements early in the system design process. UMLsec has been integrated with CASE tools that support automated analysis routines in order to verify the models developed with UMLsec against security requirements by utilizing the constraints associated with the UMLsec stereotypes [12]. Early investigations to generate annotations from UMLsec specifications and to automatically verify code against these annotations were discussed in [13]. Further UMLec traceability techniques based on refactoring are discussed in [14]. The techniques presented by Yu et al. trace mismatch between cryptographic implementation code and the UMLsec model. In contrast, SecureUML proposes the use of Model Driven Architecture (MDA) to generate the target system architecture from the system models, which is referred to as *Model Driven Security* [11]. The idea of executable models bridges the gap that exists between the design and actual implementation. As noted previously, this gap provides an avenue through which defects can be introduced and, ultimately, become vulnerabilities within the production environment. As an extension to UML, USP is not yet fully mature, although future work looks to either exporting the USP model as part of software implementation or other types of formal analysis for verification purposes [6].

All three of the extensions to UML discussed have not yet incorporated some of the new concepts for modeling that are now provided in UML 2. Some of these features, such as capsules and ports, have the potential to provide additional support in the modeling of security protocols.

The strengths of approaches that utilize and extend UML is their foundation in a common, de-facto standard for object-oriented modeling that has been adopted within industry. However, USP and UMLsec do not provide the rigorous, traceable and automatic transition from design to implementation code. Only SecureUML, utilizes the concept of Model Driven Development (MDD) but it does not use any specific toolset in the generation of code from design and relies upon the systems architect to construct the transformation functions. The weakness in this approach is the ability to verify that the transformation functions maintain the correctness of the security design language used to construct the models. In addition, the focus of SecureUML is on Role-Based Access Control (RBAC) and it is not specifically applied to security protocols.

### 3.4 Conclusion on the Related Work

From our review and discussion of the related work based on our modeling criteria, we can see that the current modeling approaches discussed in this section fail to address all the essential modeling requirements elaborated earlier. Also a significant semantic gap exists between model and product in the three approaches discussed above. We propose to meet the essential requirements and reduce this semantic gap by using the strength of MDD and UML2.

## 4 UML 2 Real-time Features

Model Driven Development (MDD) evolved in the early 1990's in the telecommunication and telephony industry. UML 2 implements the concepts of MDD. Although several tools on the market claim full or partial compliance with UML 2, we focus on the design concepts and engineering methodology provided by the Real-Time Object-Oriented Modeling (ROOM) as first introduced by Selic et al. [15]. As opposed to common engineering, an MDD design model is not only an artifact of engineering but it also generates the final product through code generation. This translation of design to code allows the model to become the end product; hence maintenance of the model allows the code to be maintained in parallel, models are no longer throwaways [16]. Our choice of using MDD and UML 2 to model communication protocols is based on strong encapsulation, bounded state behavior and the reduced semantic gap provided by the easy, traceable and rigorous translation of models to products. UML 2 provides several features that will allow us to meet our modeling criteria:

- Strong encapsulation through the use of capsules and typed ports. Capsules are Object Oriented (OO) classes that can only communicate through typed ports limiting communications between participants;
- Nested capsules that provide strong containment to create independently deployable components. A top level capsule is the container that identifies the code to be automatically generated. Nested capsules allow the designer to separate the structure of each participant in a security protocol and independently model their behavior;

- Each port is an instantiation of a protocol class that specifically identifies in and out messages that can either be sent or received depending on port direction (base/client or conjugate/server). As such protocols and ports provide for stronger encapsulation than that found in UML 1.X; by limiting communications between classes that contain defined ports;
- Nested state machines with run to completion semantic. Each layer of a state machine allows for separation of concern between each mode and state in the protocol enabling simple behavioral design. The rigor of Finite State Machine with stable states provide bounded and predictable behaviour;
- Automatic code generation provides for easy and reliable code translation from designs; and
- Executable designs with state monitoring, trace capture and sequence diagrams. Toolsets that fully support UML 2 provide target observability during design/code execution. Executable designs and state monitoring provide rigor for the verification and validation of security protocols. Sequence diagrams can then be provided as validated protocol specifications.

The strength of MDD is to model highly concurrent processes that are state-based and which communicate through well defined messages. In our review of the literature we found no references to protocol conformance analysis methodologies or tools that will allow a communication security protocol designer to test his/her protocol. We believe that the behavior of communication protocols can be tested by modeling them using UML 2. UML 2 provides the capability to execute and visualize system behavior from design models. The ability to extract sequence diagrams from executable models will allow us to verify and validate working or broken security protocols in an environment where:

- all participants are well behaved;
- one or more participant(s) is not well behaved through omission failure;
- one or more participant(s) is not well behaved mischievously; and
- arbitrary failure occurs.

## 5 Initial Modeling Experiences

The approach being undertaken is to develop a modeling technique using UML 2, without extensions, taking advantage of the new features within this version that support the definition of communication between participants and provides the ability to create an executable model. Utilizing UML with no extensions allows us to meet several of our essential criteria, outlined earlier in the paper, on what is required to ensure that a modeling language can successfully be applied to engineer software security. By using UML 2 and its MDD features we can leverage UML as a de-facto standard in order to ensure that the modeling language is easy to learn and not a barrier to adoption and that the models can bridge the current gap that exists between the design and implementation. As part of our modeling exercise we intend to verify that the security protocols modeled in UML 2 are expressive enough to present the key concepts of a security protocol without losing any important details and that the executable models are predictive in that they reveal any non-obvious properties of the security protocol being modeled.

In order to validate the security protocol models created using this approach we will initially model a security protocol with known flaws in order to determine if the expected behaviours are revealed within the model. Initially we will apply an informal visual inspection approach of the model. Then we will execute the model and from this execution examine the state monitor, sequence diagrams and utilize the trace capture in order to examine these artefacts to verify if the expected behaviours of the procotol are evident during execution. We will then model a security protocol with no known flaws and apply the approach described above in order to validate if the model is expressive and predictive. We will then model several other security protocols and adjust our modeling technique from findings. For all security protocol models created, we will subject them to adversial behaviours with the intent that this will reveal any deficiencies in the design or the actual model itself. By comparing what the visual model reveals about the security protocol with the actual behaviours exposed during execution we hope to determine if there are any deficiencies with the approach of using UML 2, with no extensions, for modeling security protocols.

Our approach in the development of the design model for a security protocol includes the modeling of the key principals A, B and the server. Each is defined as a capsule that has a set of ports, where these ports provide the interface by which principals can communicate with one another without one principal requiring knowledge about the internal specification of another principal. To dictate the set of allowable messages that can be exchanged between the principals a set of protocols are defined and associated to the ports on each capsule. State machines define the ordering of the messages within the protocol in order to establish the correct sequence of allowable interactions between the principals, as defined by the security protocol.

For all of the principals there is a capsule that defines the general structure of what constitutes a principal. From this class, the principals A, B and the Adversary are derived. The Server is treated as a separate entity and is defined as a separate capsule.

In our experimentation we have found that one of the key aspects of the model is the role of the Adversary within the context of a security protocol and so some fundamental security assumptions need to be made:

- The Adversary is able to eavesdrop on all security protocols;
- The Adversary has the ability to alter all of the messages sent using any information available, reroute any message to another principal, and generate and insert completely new messages; and
- The Adversary can be a legitimate protocol participant (insider) or an external party (outsider) or a combination of the two [17].

The security assumptions made within the design follow the Dolev-Yao threat model where it is assumed that the open communication network is under the control of the adversary in monitoring, intercepting, rerouting and inserting new messages [18]. To achieve the security assumptions as described above the design needs to ensure that the Adversary in the model has the ability to eavesdrop on all messages whether or not the adversary choses to act on any of those messages. This is addressed in the design by encapsulating the principals, A and B, within the Adversary as illustrated in Figure 1. The UML 2 diagram in Figure 1 has been simplified and is shown without the details of the capsules for clarity. The Adversary routes all messages between the principals, A, B and the Server. This design supports the security assumptions that are made about the role of the Adversary.

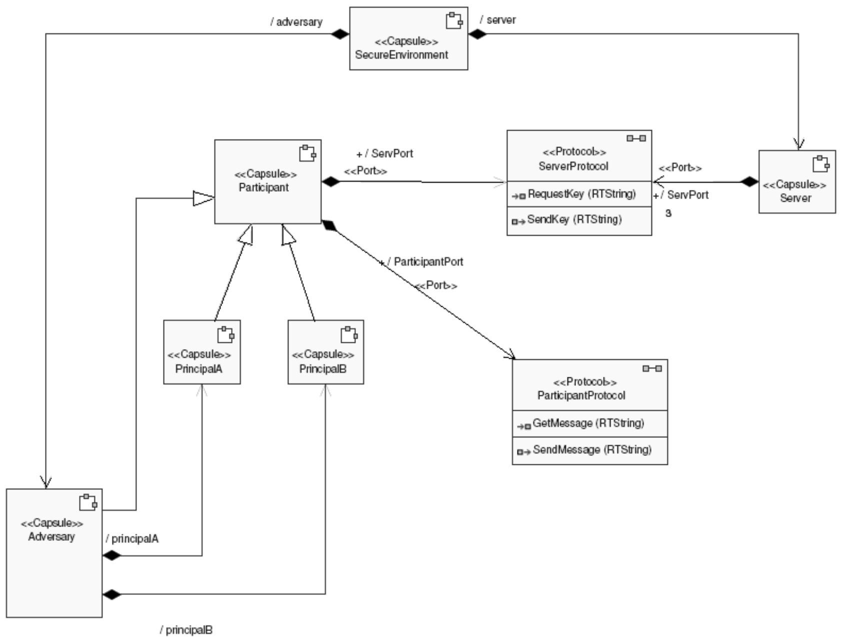

**Fig. 1.** Class Diagram of Participants in a Security Protocol

## 6 Conclusion

We have completed a critical review of the state of the art in modeling and validating security protocols. We have elaborated a set of requirements for modeling security protocols that allows for injection of threats from potential adversaries to verify the model against vulnerabilities. Our initial modeling efforts using UML 2 show a significant potential for this modeling language to be used without modifications, add-ons or extensions to fully model and implement communication protocols.

## References

1. Group, O.M.: Unified modeling language 2. Technical Report UML 2.1.2, OMG-UML (November 2007)
2. Devanbu, P.T., Stubblebine, S.: Software engineering for security: A roadmap. ICES 2000 special volume on the Future of Software Engineering (2000)
3. Jürjens, J.: Formal development and verification of security-critical systems with UML (position paper)
4. Jürjens, J.: UMLsec: Extending UML for secure systems development. Lecture Notes in Computer Science **2460** (2002) 412–425
5. Ryan, P., Schneider, S., Goldsmith, M., Lowe, K., Roscoe, B.: The Modelling and Analysis of Security Protocols: The CSP Approach. Addison-Wesley (2001) ISBN: 0 201 67471 8.
6. Zhitang Li, Y.X., Li, W.: USP: Modeling security protocol with UML. Network Architectures, Management and Applications IV (2006)
7. Bobkowska, A.: A framework for methodologies of visual modeling language evaluation. In Proceedings of the 2005 Symposia on Metainformatics MIS '05 **214** (November 2005)
8. Selic, B.: The pragmatics of model-driven development. IEEE Software **20**(5) (September 2003) 19–25
9. Schneider, S.: Security properties and CSP. IEEE Computer Society Symposium on Security and Privacy (1996)
10. McDermott, J.: Visual security protocol modeling. Proceedings of the 2005 workshop on New Security Paradigms (2005) 97–109
11. Basin, D., Doser, J., Lodderstedt, T.: Model driven security: From UML models to access control infrastructures. ACM Transactions on Software Engineering Methodology **15**(1) (2006) 39–91
12. Jürjens, J.: Sound methods and effective tools for model-based security engineering with UML. (2005) 322–331
13. Jürjens, J.: Security analysis of crypto-based Java programs using automated theorem provers. In: 21st International Conference on Automated Software Engineering (ASE 2006), IEEE/ACM (2006) 167–176
14. Yu, Y., Jürjens, J., Mylopoulos, J.: Application of traceability to maintenance of secure software. In: (To Appear): 24th IEEE International Conference on Software Maintenance (ICSM), IEEE (2008) 167–176
15. Selic, B., Gullekson, G., Ward, P.: Real-Time Object-Oriented Modeling. John Wiley & Sons (1994) ISBN: 0 471 59917 4.
16. France, R.B., Ghosh, S., Trong, T.D., Solberg, A.: Model driven development using uml 2.0: Promises and pitfalls. IEEE Computer **39** (February 2006) 59–66
17. Boyd, C., Mathuria, A.: Protocols for Authentication and Key Establishment. Springer Verlag (2003)
18. Mao, W.: A structured operational semantic modelling of the Dolev-Yao threat environment and its composition with cryptographic protocols. Computer Standards and Interfaces **27** (June 2005) 479–488